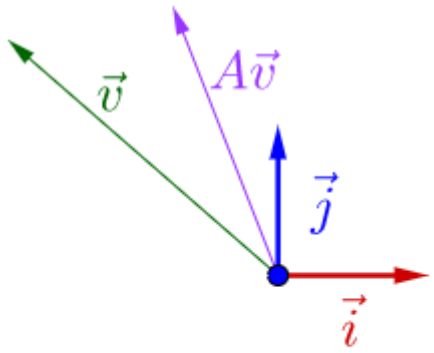
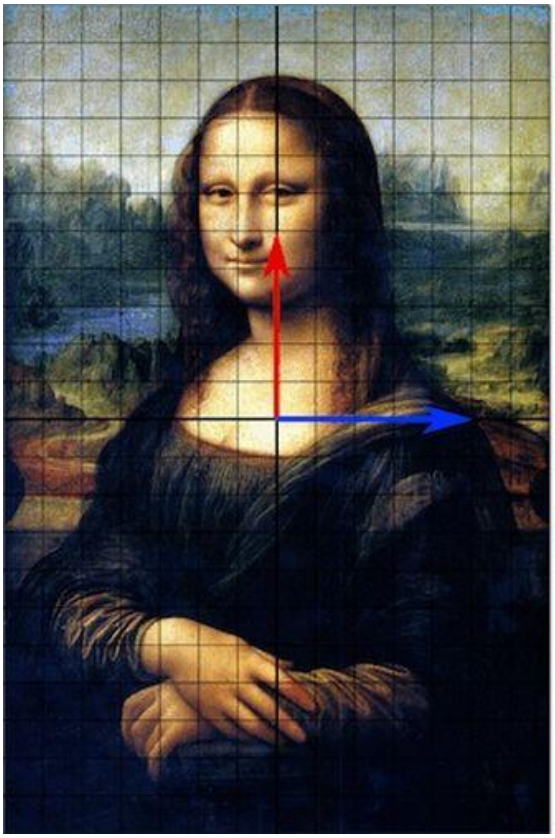
1、特征值和特征向量

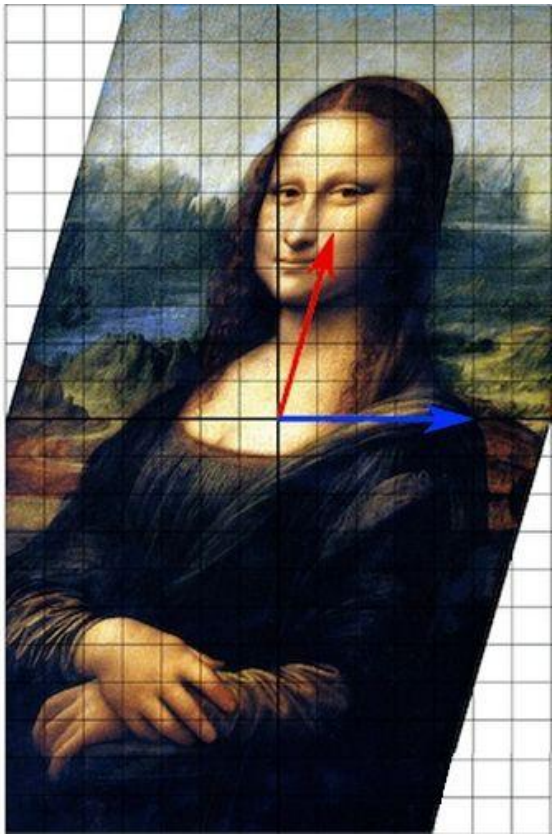
1.1、概念定义

两个向量相乘效果图如下：

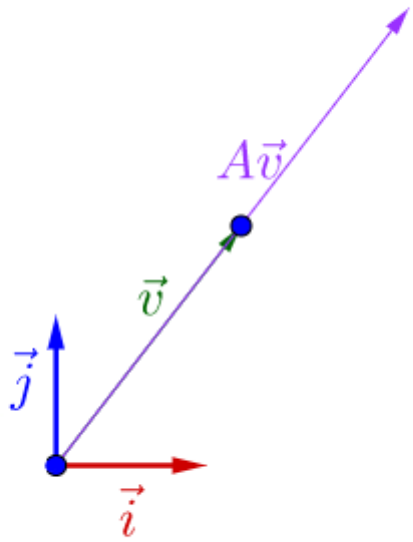


看图，这张图片竖直方向进行偏移，形象说明了向量相乘效果：





设 A 是 n 阶方阵，如果存在数 λ 和非零 n 维**列向量** v，使得 $A\vec{v}=\lambda\vec{v}$ 成立，则称 λ 是矩阵 A 的一个特征值(eigenvalue)，$\vec{v}$ 是特征值 λ 对应的特征向量（eigenvector）。



矩阵 A 对向量 $\vec{v}$ 进行变换，这个变换的特殊之处是当它作用在特征向量 $\vec{v}$ 上的时候，$\vec{v}$ 只发生了缩放变换，它的方向并没有改变，并没有旋转。

观察发现，$\vec{v}$ 和 $A\vec{v}$ 在同一条直线上，只是长度不同，此时我们称 $\vec{v}$ 是 A 的特征向量，而 $A\vec{v}$ 的长度是 $\vec{v}$ 长度的 λ 倍，λ 就是特征值。

如果n阶方阵 A 是满秩矩阵，那么矩阵 A 有n个不同的特征值和特征向量。

```
import numpy as np
A = np.random.randint(1,10,size = (4,4))
display(A)
print('------------------------')
if np.linalg.matrix_rank(A) == 4:# 必须是满秩矩阵
    # 实对称矩阵特征值为实数,非对称矩阵和复矩阵特征值可能为复数
    w,v = np.linalg.eig(A) # 返回特征值和特征向量
    display(w,v)
    w = np.real(w)
    v = np.real(v)
    print('------------------------')
    display(A.dot(v[:,0]))
    display(w[0]*v[:,0])
'''
array([[7, 7, 1, 4],
       [9, 6, 7, 4],
       [5, 3, 5, 8],
       [4, 7, 8, 3]])
------------------------
array([21.954224  +0.j        ,  5.01185053+0.j        ,
       -2.98303727+2.43963317j, -2.98303727-2.43963317j])
array([[ 0.43707269+0.j        ,  0.64325095+0.j        ,
```

```
         0.19617028+0.39807527j,  0.19617028-0.39807527j],
       [ 0.57873235+0.j        ,  0.16678659+0.j        ,
        -0.17496565-0.51124289j, -0.17496565+0.51124289j],
       [ 0.46910247+0.j        , -0.58461391+0.j        ,
         0.45970147+0.0832835j ,  0.45970147-0.0832835j ],
       [ 0.5039635 +0.j        , -0.46544282+0.j        ,
        -0.5411187 +0.j        , -0.5411187 -0.j        ]])
------------------------
array([ 9.5955917 , 12.70561958, 10.29878082, 11.06412746])
array([ 9.5955917 , 12.70561958, 10.29878082, 11.06412746])
'''
```

实对称矩阵，求解特征值和特征向量为实数

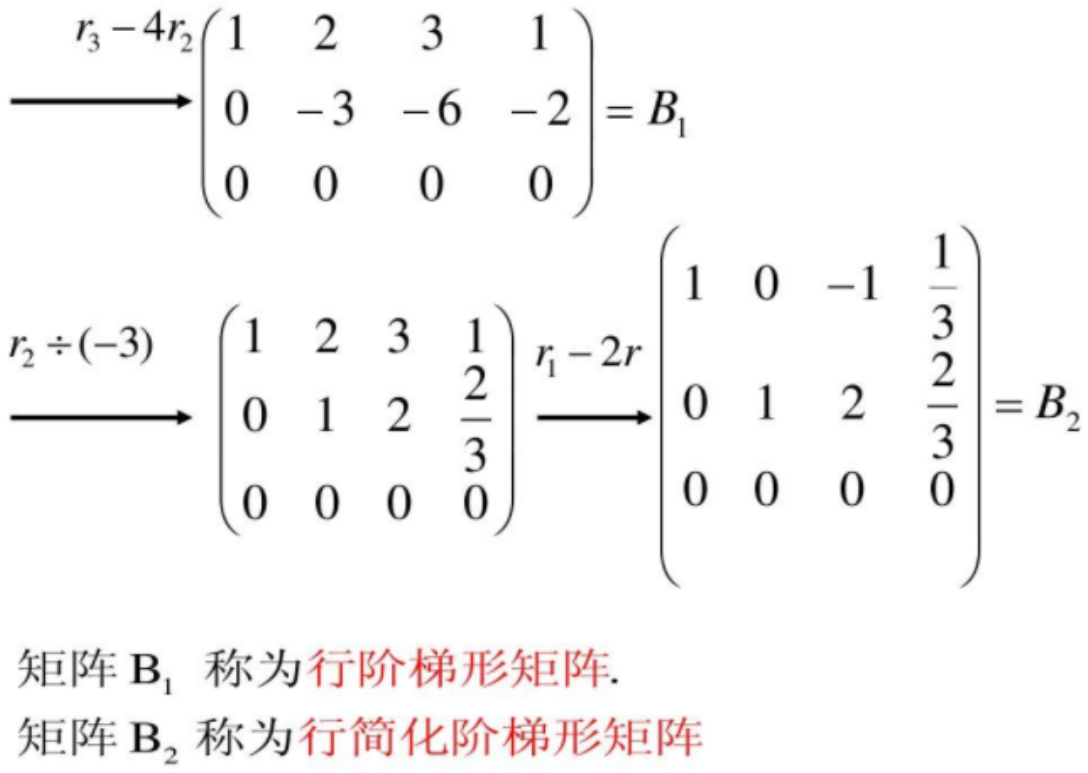
```
# 实对称矩阵
import numpy as np
B = np.array([[1,2,3],[2,5,8],[3,8,9]])
np.linalg.eig(B)
'''
(array([16.10241769,  0.19192041, -1.2943381 ]),
 array([[ 0.23165721,  0.95684242, -0.17546374],
        [ 0.59582744, -0.28213926, -0.75192227],
        [ 0.76897633, -0.0696421 ,  0.63547256]]))
'''
```

矩阵的秩: 用初等行变换将矩阵 A 化为阶梯形矩阵, 则矩阵中非零行的个数就定义为这个矩阵的秩, 记为 $r(A)$ 。

初等行变换

$$\begin{cases} x_1+2x_2+3x_3=1 \\ 4x_1+5x_2+6x_3=2 \\ 7x_1+8x_2+9x_3=3 \end{cases} \qquad B=\begin{pmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 2 \\ 7 & 8 & 9 & 3 \end{pmatrix}$$

$$B=\begin{pmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 2 \\ 7 & 8 & 9 & 3 \end{pmatrix} \xrightarrow[r_3-7r_1]{r_2-4r_1} \begin{pmatrix} 1 & 2 & 3 & 1 \\ 0 & -3 & -6 & -2 \\ 0 & -6 & -12 & -4 \end{pmatrix}$$

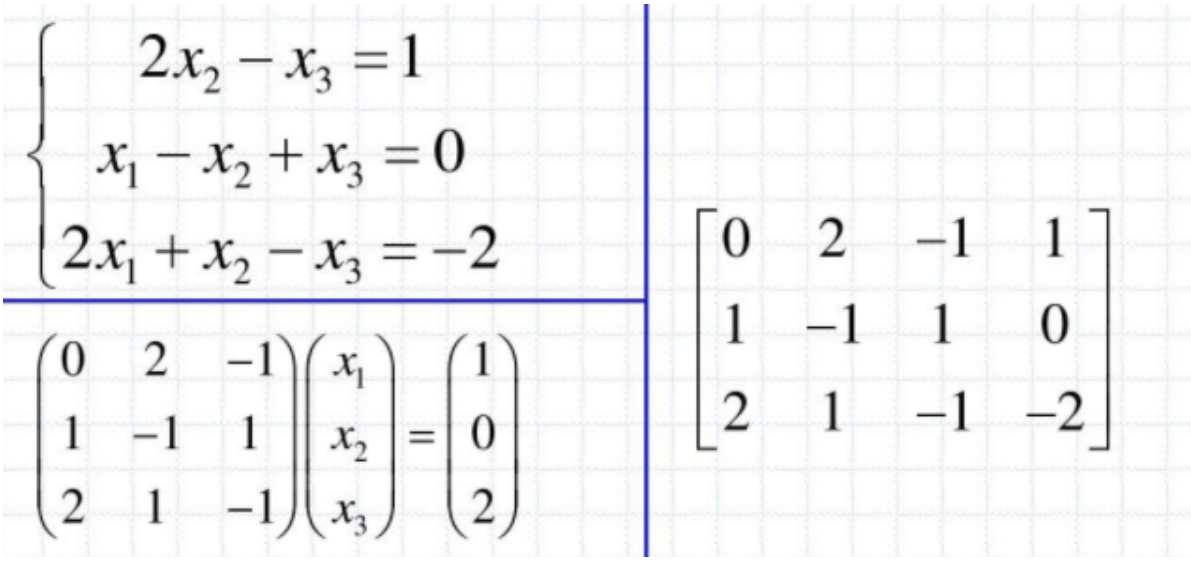


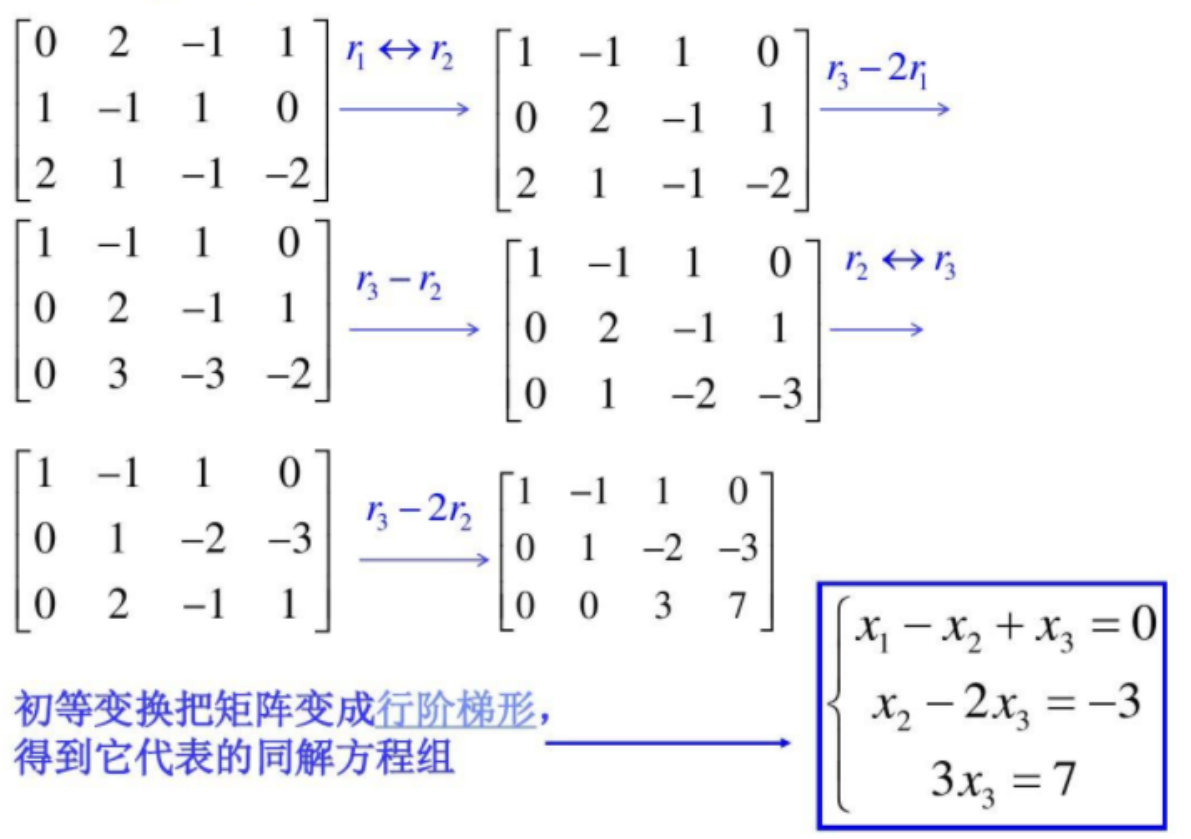
经过初等变换可知这个方程无解。

```
import numpy as np
X = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
y = np.array([1,2,3])

# 查看矩阵秩，输出结果为2
print('矩阵X的秩为：',np.linalg.matrix_rank(X))

# 尝试求解报错，说明方程没有唯一解
np.linalg.solve(X,y)
```

1.2、满秩矩阵

满秩矩阵（non-singular matrix）：设 A 是 n 阶矩阵, 若 $r(A)=n$, 则称 A 为**满秩矩阵**。但满秩不局限于 n 阶矩阵。若矩阵秩等于行数，称为行满秩；若矩阵秩等于列数，称为列满秩。既是行满秩又是列满秩则为 n 阶矩阵即 **n 阶方阵**。





根据初等行变换，可得：

$x_1 = -\frac{2}{3} = -0.67$

$x_2 = \frac{5}{3} \approx 1.67$

$x_3 = \frac{7}{3} \approx 2.33$

1.3、方程求解

```
import numpy as np
X = np.array([[0,2,-1],
              [1,-1,1],
              [2,1,-1]])
y = np.array([1,0,-2])
```

```
# 查看矩阵秩
print('矩阵X的秩为：',np.linalg.matrix_rank(X))

# 尝试求解报错，说明方程没有唯一解
np.linalg.solve(X,y).round(2)
'''
矩阵X的秩为： 3
array([-0.67,  1.67,  2.33])
'''
```

1.4、特征值和特征向量示例

所有特征值的乘积等于 A 的行列式的：

$$\prod_{i=1}^{n} \lambda_i = |A|$$

```
import numpy as np
X = np.array([[2,3,7],
              [1,5,8],
              [0,4,9]])
# w表示特征值，v表示特征向量
w,v = np.linalg.eig(X)
print('矩阵X的行列式：',np.linalg.det(X))
print('特征值累乘值：',np.round(np.real(np.prod(w))))
'''
矩阵X的行列式： 27.0
特征值累乘值： 27.0
'''
```

特征值和特征向量在机器学习中会被用到，像 PCA 主成分分析，LDA 线性判别分析，以及其它算法里面都会用到它的理论和方法。

2、特征值分解

2.1、特征值分解定义与操作

特征值分解，就是将矩阵A分解为如下式：

$$A = Q\Sigma Q^{-1}$$

其中，Q 是矩阵 A 的**特征向量**组成的矩阵，Σ 则是一个**对角阵**，对角线上的元素就是**特征值。**

$$\Sigma = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}$$

```
import numpy as np
A = np.array([[7, 8, 4, 3],
              [2, 9, 6, 8],
              [1, 6, 9, 6],
              [7, 9, 7, 3]])
# w（Σ）表示特征值，v（Q）表示特征向量
w,v = np.linalg.eig(A)
print('矩阵A的特征值和特征向量：')
display(w,v)

# 根据特征值分解公式可得
print('特征值和特征向量运算反推矩阵A：')
display(v.dot(np.diag(w)).dot(np.linalg.inv(v)))
'''
矩阵A的特征值和特征向量：
array([23.92597521, -2.03307797,  4.89283825,  1.21426451])
array([[ 0.45784163, -0.28165575,  0.69443965,  0.62434124],
       [ 0.53567307,  0.53012742,  0.03104449, -0.66275073],
       [ 0.46235516,  0.16303348, -0.65314778,  0.12742158],
       [ 0.5382036 , -0.78297832,  0.30031282,  0.39334874]])
特征值和特征向量运算反推矩阵A：
array([[7., 8., 4., 3.],
       [2., 9., 6., 8.],
       [1., 6., 9., 6.],
       [7., 9., 7., 3.]])
'''
```

如果矩阵 A 是对称矩阵，那么 Q 是正交矩阵，正交矩阵的定义是 Q 的逆等于 Q 的转置

$$Q^{-1} = Q^T$$

代码验证：

```
# 实对称矩阵
import numpy as np
B = np.array([[1,2,3],
              [2,5,8],
              [3,8,9]])
# w（Σ）表示特征值，v（Q）表示特征向量
w,v = np.linalg.eig(B)
print('矩阵A的特征值和特征向量：')
display(w,v)
print('特征值和特征向量运算反推矩阵A：')
display(v.dot(np.diag(w)).dot(np.linalg.inv(v)))
display(v.dot(np.diag(w)).dot(v.T))
print('Q是正交矩阵：')
display(np.linalg.inv(v),v.T)
'''
矩阵A的特征值和特征向量：
array([16.10241769,  0.19192041, -1.2943381 ])
array([[ 0.23165721,  0.95684242, -0.17546374],
       [ 0.59582744, -0.28213926, -0.75192227],
```

```
       [ 0.76897633, -0.0696421 ,  0.63547256]])
特征值和特征向量运算反推矩阵A：
array([[1., 2., 3.],
       [2., 5., 8.],
       [3., 8., 9.]])
array([[1., 2., 3.],
       [2., 5., 8.],
       [3., 8., 9.]])
Q是正交矩阵：
array([[ 0.23165721,  0.59582744,  0.76897633],
       [ 0.95684242, -0.28213926, -0.0696421 ],
       [-0.17546374, -0.75192227,  0.63547256]])
array([[ 0.23165721,  0.59582744,  0.76897633],
       [ 0.95684242, -0.28213926, -0.0696421 ],
       [-0.17546374, -0.75192227,  0.63547256]])
'''
```

2.2、特征值分解意义

一个矩阵其实就是一个**线性变换**，因为一个矩阵乘以一个向量后得到的向量，其实就相当于将这个向量进行了线性变换。

当矩阵是高维的情况下，那么这个矩阵就是高维空间下的一个线性变换，这个线性变化可能没法通过图片来表

示，但是可以想象，这个变换也同样有很多的变换方向，我们通过特征值分解得到的前 N 大特征向量，那么就对

应了这个矩阵最主要的 N 个变化方向。我们利用这前 N 个变化方向，就可以近似表达这个矩阵（变换）。也就是

说的：**提取**这个矩阵**最重要**的特征。

总结一下，特征值分解可以得到特征值与特征向量，特征值**大小**表示的是这个特征到底有多**重要**，而特征向量表示这个特征**是什么**，可以将每一个特征向量理解为一个线性的子空间，我们可以利用这些线性的子空间干很多的事情。

不过，特征值分解也有很多的**限制**，比如说变换的矩阵必须是**方阵**。

3、矩阵和向量求导公式

3.1、常见矩阵求导公式

有六种矩阵求导公式如下：

类型	标量（Scale）	向量 （Vector）	矩阵（Matrix）
标量（Scale）	$\frac{\partial y}{\partial x}$	$\frac{\partial \vec{y}}{\partial x}$	$\frac{\partial Y}{\partial x}$
向量 （Vector）	$\frac{\partial y}{\partial \vec{x}}$	$\frac{\partial \vec{y}}{\partial \vec{x}}$	$\frac{\partial Y}{\partial \vec{x}}$
矩阵（Matrix）	$\frac{\partial y}{\partial X}$	$\frac{\partial Y}{\partial X}$	$\frac{\partial Y}{\partial X}$

3.2、向量求导公式

带横线的小写字母 $\vec{a}$ 表示列向量，大写字母 A 表示矩阵， $\vec{a}$和 A 都不是 $\vec{x}$ 的函数。

$$\frac{\partial \vec{a}^T\vec{x}}{\partial \vec{x}} = \frac{\partial \vec{x}^T\vec{a}}{\partial \vec{x}} = \vec{a}^T$$

$$\frac{\partial \vec{x}^T\vec{x}}{\partial \vec{x}} = 2\vec{x}^T$$

$$\frac{\partial A\vec{x}}{\partial \vec{x}} = A$$

$$\frac{\partial \vec{x}^T A}{\partial \vec{x}} = A^T$$

$$\frac{\partial \vec{x}^T A\vec{x}}{\partial \vec{x}} = (A + A^T)\vec{x}$$

$$\frac{\partial (\vec{x}^T\vec{a})^2}{\partial \vec{x}} = 2\vec{x}^T\vec{a}\vec{a}^T$$

3.3、矩阵求导公式

$$\frac{\partial X^T}{\partial X} = \frac{\partial X}{\partial X^T} = I$$

$$\frac{\partial AX}{\partial X} = \frac{\partial XA}{\partial X} = A^T$$

$$\frac{\partial AX^T}{\partial X} = \frac{\partial X^T A}{\partial X} = A$$

$$\frac{\partial X^T AX}{\partial X} = (A + A^T)X$$ ； A不是对称矩阵

$$\frac{\partial X^T AX}{\partial X} = 2AX$$ ； A是对称矩阵

转置公式如下：

- $(mA)^T = mA^T$，其中m是常数
- $(A+B)^T = A^T + B^T$
- $(AB)^T = B^T A^T$
- $(A^T)^T = A$

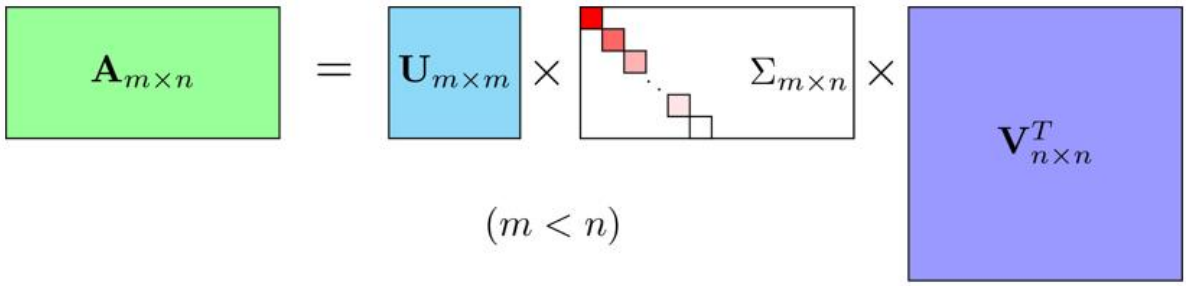
4、奇异值分解（SVD）

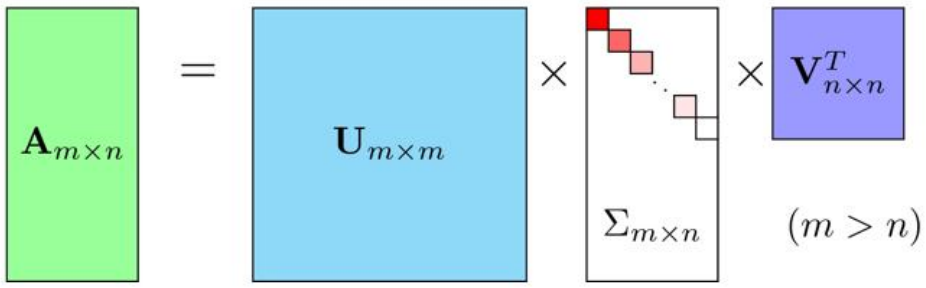
4.1、什么是奇异值分解

特征值分解是一个提取矩阵特征很不错的方法，但是它**只适用于方阵**。而在现实的世界中，我们看到的大部分矩阵都不是方阵，比如说有 m 个学生，每个学生有 n 科成绩，这样形成的一个 m * n 的矩阵就可能不是方阵，我们怎样才能像描述特征值一样描述这样一般矩阵呢的重要特征呢？奇异值分解就是用来干这个事的，奇异值分解是一个能适用于任意的矩阵的一种分解的方法。

$$A = U\Sigma V^T$$

假设 A 是一个 m * n 的矩阵，那么得到的U是一个 m * m 的方阵（里面的向量是正交的，U 里面的向量称为**左奇异向量**），$\sum$ 是一个 m * n 的实数对角矩阵（对角线以外的元素都是0，对角线上的元素称为**奇异值**），V^T 是一个n * n 的矩阵，里面的向量也是正交的，V 里面的向量称为**右奇异向量**），从下图片来反映几个相乘的矩阵的大小关系：





4.2、奇异值与特征值关系

特征值分解：

$$A = Q\Sigma Q^{-1}$$

奇异值分解：

$$A = U\Sigma V^T$$

那么奇异值和特征值是怎么对应起来的呢？首先，我们将矩阵 A 的转置 和 A 做矩阵乘法，将会得到一个方阵，我们用这个方阵求特征值可以得到：

$$(A^TA)\vec{v}_i = \lambda_i\vec{v}_i$$

这里得到的 $\vec{v}$，就是我们上面的**右奇异向量。**

然后，我们将矩阵 A 和 A 的转置做矩阵乘法，将会得到一个方阵，我们用这个方阵求特征值可以得到：

$$(AA^T)\vec{u}_i = \lambda_i\vec{u}_i$$

这里得到的 $\vec{u}$，就是我们上面的**左奇异向量。**

此外我们还可以得到：

$$A = U\Sigma V^T$$

$$AV = U\Sigma V^TV$$

因为A^TA，AA^T 是对阵矩阵，所以 V,U 是正交矩阵，正交矩阵的定义是 V 的逆等于 V 的转置，即 $V^TV = V^{-1}V = I$；$U^TU = U^{-1}U = I$。

$$AV = U\Sigma$$

$$A\vec{v}_i = \sigma_i\vec{u}_i$$

$$\sigma_i = \frac{|A\vec{v}_i|}{|\vec{u}_i|}$$

这里的 σ_i 就是上面说的**奇异值。**

存在的问题?

对 AA^T 以及 A^TA 做**特征值分解**，即可得到奇异值分解的结果。但是样分开求存在一定的问题，由于做特征值分解的时候，特征向量的正负号并不影响结果，比如：

$$AA^T\vec{u}_i = \sigma_i\vec{u}_i \quad or \quad AA^T(-\vec{u}_i) = \sigma_i(-\vec{u}_i)$$

$$A^TA\vec{v}_i = \sigma_i\vec{v}_i \quad or \quad A^TA(-\vec{v}_i) = \sigma_i(-\vec{v}_i)$$

如果在计算过程取，取上面的 $\vec{u}_i$ 组成左奇异矩阵 U，取 $-\vec{v}_i$ 组成右奇异矩阵 V，此时 $A \neq U\Sigma V^T$。因此求 $\vec{v}_i$ 时，要根据 $\vec{u}_i$ 来求，这样才能保证 $A = U\Sigma V^T$。

解决方案:

1、计算特征值:

$$A = U\Sigma V^T$$

$AA^T = U\Sigma V^TV\Sigma^TU^T = U\Sigma\Sigma^TU^T$ 根据结合律$V^TV = I$，因此可以化简

可以得到左奇异矩阵 $U \in R^{m \times m}$ 和奇异值矩阵 $\Sigma \in R^{m \times m}$，根据上面分解公式可知，奇异值矩阵为特征值开平方！

2、间接求右奇异矩阵：求 $V \in R^{m \times n}$

因为：$A = U\Sigma V$

所以：

$(U\Sigma)^{-1}A = (U\Sigma)^{-1}U\Sigma V$

$V = (U\Sigma)^{-1}A$

$V = \Sigma^{-1}U^{-1}A$

$V = \Sigma^{-1}U^{T}A$，因为 U 是正交矩阵，所以 $U^{-1} = U^{T}$

5、求解奇异值分解

5.1、方式一

根据上面对应公式，进行奇异值分解

```
import numpy as np
A = np.array([[ 3,  4,  5,  5],
              [ 7,  5,  3,  6],
              [ 6,  5,  7,  7],
              [ 4,  9,  8,  9],
              [ 5, 10,  5,  7]])
# 右奇异矩阵
sigma,U = np.linalg.eig(A.dot(A.T))
# 降序排列后，逆序输出
index = np.argsort(sigma)[::-1]
# 将特征值对应的特征向量也对应排好序
sigma = sigma[index]
U = U[:,index]
print('右奇异矩阵如下：')
display(U)

# 计算奇异值矩阵的逆
sigma = np.sqrt([s if s >0 else 0 for s in sigma])
print('奇异值为：')
display(sigma[:4])

# 计算右奇异矩阵
sigma_inv = np.diag([1/s if s >0 else 0 for s in sigma])
V = sigma_inv.dot((U.T).dot(A))
print('左奇异矩阵如下：')
display(V[:4])

print('使用奇异值分解还原矩阵A：')
U.dot(np.diag(sigma)).dot(V).round(0)

'''
右奇异矩阵如下：
array([[-0.3101916 ,  0.04769317, -0.35708497,  0.01791768, -0.87958844],
       [-0.37495544, -0.73958189,  0.33372775, -0.44531378, -0.0524258 ],
       [-0.44812844, -0.34942375, -0.5206661 ,  0.52493567,  0.36115552],
       [-0.55774569,  0.50103456, -0.20813073, -0.55345231,  0.29707954],
```

```
       [-0.50128858,  0.27858822,  0.66835958,  0.46851647, -0.06990107]])
奇异值为：
array([27.56758049,  4.36911093,  3.79131357,  0.75187562])
左奇异矩阵如下：
array([[-0.39834686, -0.55822029, -0.46362907, -0.56103295],
       [-0.85451182,  0.46712883,  0.223151  , -0.04247162],
       [ 0.17147559,  0.64552498, -0.72590995, -0.16415961],
       [ 0.28587663,  0.23125369,  0.45640603, -0.81024059]])
使用奇异值分解还原矩阵A：
array([[ 3.,  4.,  5.,  5.],
       [ 7.,  5.,  3.,  6.],
       [ 6.,  5.,  7.,  7.],
       [ 4.,  9.,  8.,  9.],
       [ 5., 10.,  5.,  7.]])
'''
```

5.2、方式二

根据，NumPy提供的方法，进行奇异值求解

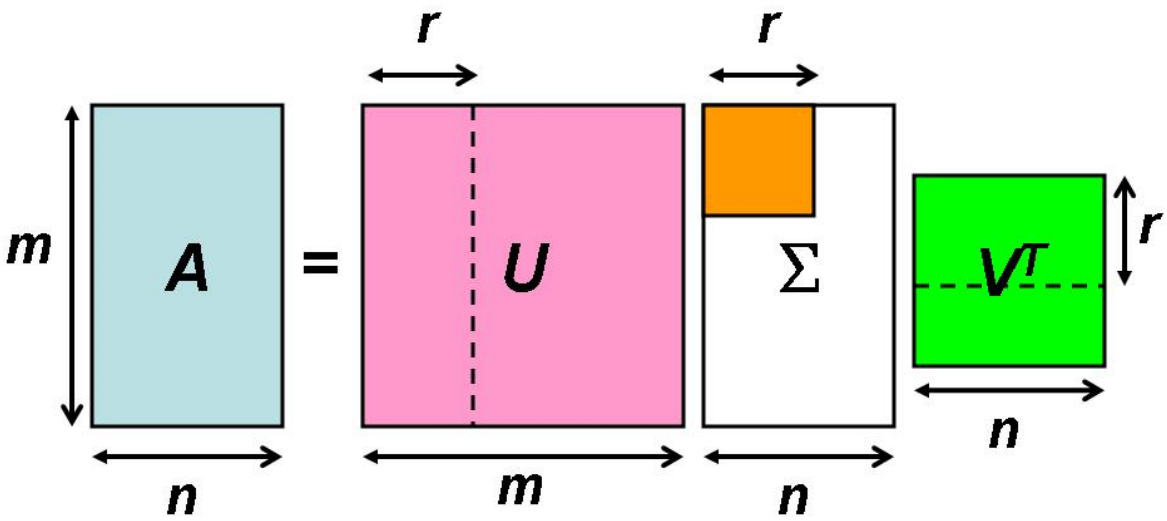
```
import numpy as np
A = np.array([[ 3,  4,  5,  5],
              [ 7,  5,  3,  6],
              [ 6,  5,  7,  7],
              [ 4,  9,  8,  9],
              [ 5, 10,  5,  7]])
u,s,v =np.linalg.svd(A)
display(u,s,v)
'''
array([[-0.3101916 ,  0.04769317, -0.35708497, -0.01791768,  0.87958844],
       [-0.37495544, -0.73958189,  0.33372775,  0.44531378,  0.0524258 ],
       [-0.44812844, -0.34942375, -0.5206661 , -0.52493567, -0.36115552],
       [-0.55774569,  0.50103456, -0.20813073,  0.55345231, -0.29707954],
       [-0.50128858,  0.27858822,  0.66835958, -0.46851647,  0.06990107]])
array([27.56758049,  4.36911093,  3.79131357,  0.75187562])
array([[-0.39834686, -0.55822029, -0.46362907, -0.56103295],
       [-0.85451182,  0.46712883,  0.223151  , -0.04247162],
       [ 0.17147559,  0.64552498, -0.72590995, -0.16415961],
       [-0.28587663, -0.23125369, -0.45640603,  0.81024059]])
'''
```

6、奇异值分解性质

奇异值 σ_i 跟特征值类似，在矩阵 $\sum$ 中也是从大到小排列，而且 σ 的减小特别的快，在很多情况下，前10%甚至1%的奇异值的和就占了全部的奇异值之和的99%以上了。也就是说，我们也可以用前 r 大的奇异值来近似描述矩阵，这里定义一下部分奇异值分解：

$$A_{m\times n} \approx U_{m\times r} \sum_{r\times r} V^T_{r\times n}$$

r 是一个远小于m、n的数。这样的矩阵乘法表示如下：



右边的三个矩阵相乘的结果将会是一个接近于 A 的矩阵，在这儿，r 越接近于 n，则相乘的结果越接近于A。而这三个矩阵的面积之和（在存储观点来说，矩阵面积越小，存储量就越小）要远远小于原始的矩阵 A，我们如果想要**压缩**空间来表示原矩阵 A，我们存下这里的三个矩阵：U、Σ、V就好了。

说句大白话，称作「**奇异值**」可能无法顾名思义迅速理解其本质，那咱们换个说法，称作「**主特征值**」，你可能就迅速了然了。

对于**非奇异**（满秩）矩阵，对应着**特征值**；对于**奇异**矩阵，就需要进行奇异值分解，对应着奇异值。对于奇异矩阵，将A与其转置相乘 A^TA 将会得到一个方阵，再求特征值。值得注意的是，对于**非奇异**矩阵进行奇异值分解（SVD），得到的奇异值，其实就是特征值。

```
import numpy as np
A = np.array([[ 3,  4,  5,  5],
              [ 7,  5,  3,  6],
              [ 6,  5,  7,  7],
              [ 4,  9,  8,  9],
              [ 5, 10,  5,  7]])
u,s,v =np.linalg.svd(A)
m,n = A.shape
Σ = np.concatenate([np.diag(s),np.full(shape = (m-n,n),fill_value = 0)])
print('通过奇异值分解还原原数据：')
display(u.dot(Σ).dot(v))
# 抽取一部分奇异值，近似表示
print('用前r大的奇异值来近似描述矩阵:')
r = 2
display(u[:,:r].dot(Σ[:r,:r]).dot(v[:r]).round(1))
'''
通过奇异值分解还原原数据：
array([[ 3.,  4.,  5.,  5.],
       [ 7.,  5.,  3.,  6.],
       [ 6.,  5.,  7.,  7.],
       [ 4.,  9.,  8.,  9.],
       [ 5., 10.,  5.,  7.]])
用前r大的奇异值来近似描述矩阵:
array([[3.2, 4.9, 4. , 4.8],
       [6.9, 4.3, 4.1, 5.9],
       [6.2, 6.2, 5.4, 7. ],
       [4.3, 9.6, 7.6, 8.5],
       [4.5, 8.3, 6.7, 7.7]])
```

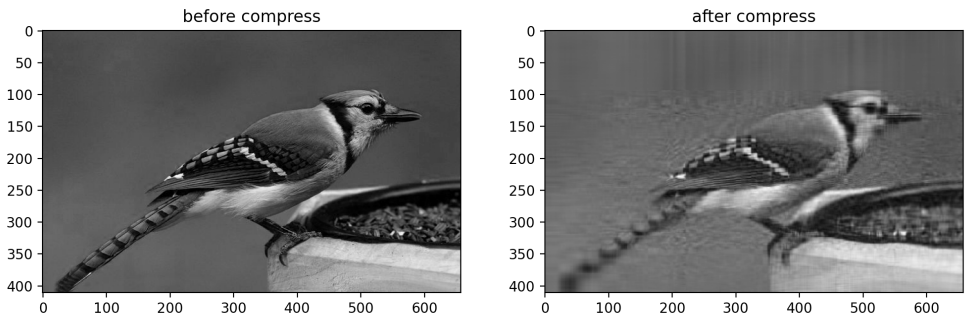
```
'''
```

7、SVD进行数据压缩

根据奇异值分解，进行数据压缩，以图片为例进行展示：

```
import numpy as np
import imageio
import matplotlib.pyplot as plt
# 图片压缩
def pic_compress(k, img):
    u, sigma, vt = np.linalg.svd(img)
    compress_img = np.dot(np.dot(u[:, :k], np.diag(sigma[:k])), vt[:k, :])  # 还
原图像
    size = u.shape[0] * k + k * k + k * vt.shape[1]  # 压缩后大小
    return compress_img, size

filename = './bird.jpg'
img = plt.imread(filename)[:,:,0]
compress_img, size = pic_compress(30, img)
print('原图尺寸:' + str(img.shape[0] * img.shape[1]))
print('压缩尺寸:' + str(size))
fig, ax = plt.subplots(1, 2,figsize = (12,4))
ax[0].imshow(img,cmap = 'gray')
ax[0].set_title('before compress')
ax[1].imshow(compress_img,cmap = 'gray')
ax[1].set_title('after compress')
plt.show()
'''
原图尺寸:270438
压缩尺寸:32970
'''
```

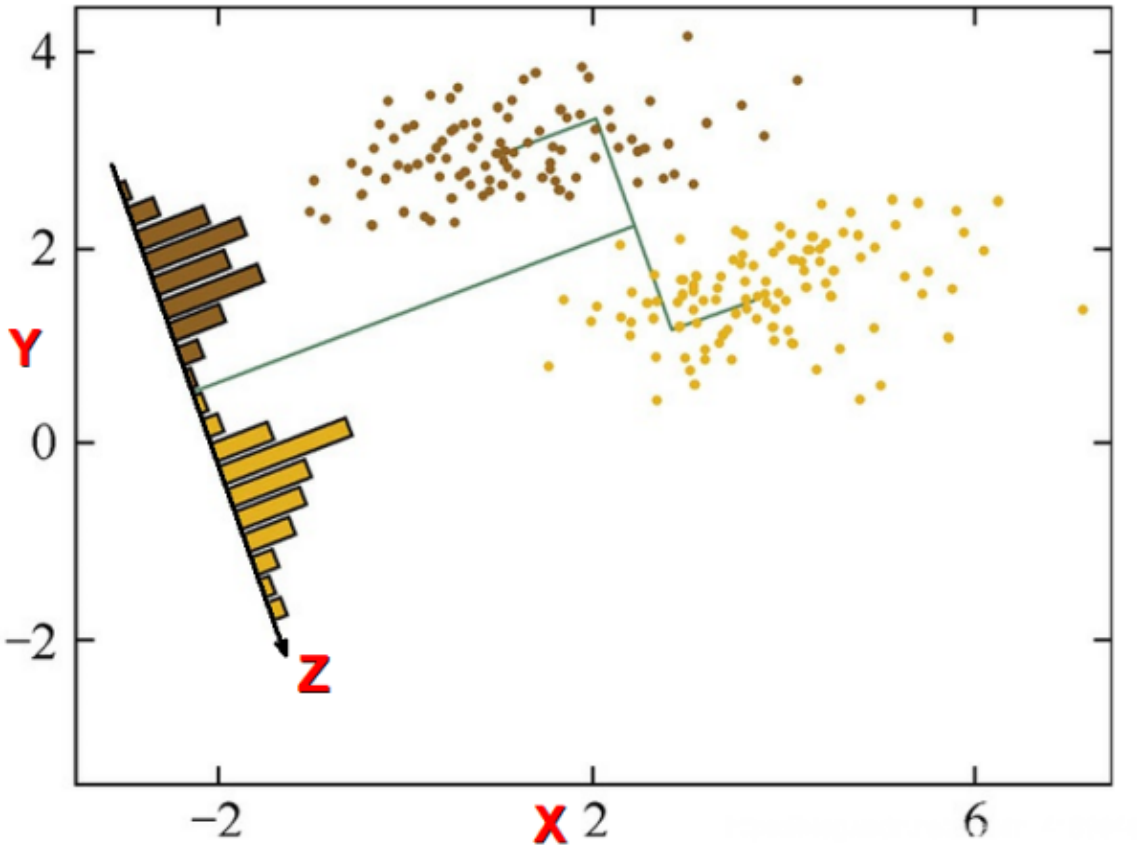
效果如下：



8、SVD进行PCA降维

8.1、什么是PCA

PCA（Principal Components Analysis）即主成分分析，是图像处理中经常用到的降维方法。它不仅仅是对高维数据进行降维，更重要的是经过降维去除了噪声，发现了数据中的模式规律。PCA把原先的 n 个特征用数目更少的 m 个特征取代，新特征是旧特征的线性组合，这些线性组合最大化样本方差，尽量使新的 m 个特征互不相关。



8.2、Scikit-learn库实现PCA降维

```
from sklearn.decomposition import PCA
from sklearn import datasets
n_components_ = 2
X,y = datasets.load_iris(return_X_y = True)
print('原始数据特征个数是：',X.shape[1])
pca = PCA(n_components = n_components_,whiten=True)
X_pca = pca.fit_transform(X,)
display(X_pca[:5])
print('经过PCA降维，数据特征个数是：',X_pca.shape[1])
'''
原始数据特征个数是： 4
array([[-1.30533786,  0.64836932],
       [-1.31993521, -0.35930856],
       [-1.40496732, -0.29424412],
       [-1.33510889, -0.64613986],
       [-1.32702321,  0.6633044 ]])
经过PCA降维，数据特征个数是： 2
'''
```

8.3、SVD进行PCA降维

```
import numpy as np
from sklearn import datasets
r = 2 # 筛选特征个数
X,y = datasets.load_iris(return_X_y = True)
print('原始数据特征个数是：',X.shape[1])
# 1、去中心化
mean_ = np.mean(X, axis=0)
X -= mean_

# 2、奇异值分解
u,s,v = np.linalg.svd(X)

# 3、符号翻转（如果为负数，那么变成正直）
max_abs_cols = np.argmax(np.abs(u), axis=0)
signs = np.sign(u[max_abs_cols, range(u.shape[1])])
u *= signs

# 4、降维特征筛选
u = u[:, :r]

# 5、归一化
u = (u - u.mean(axis = 0))/u.std(axis = 0,ddof = 1)
display(u[:5])
print('经过PCA降维，数据特征个数是：',u.shape[1])
'''
原始数据特征个数是： 4
array([[-1.30533786,  0.64836932],
       [-1.31993521, -0.35930856],
       [-1.40496732, -0.29424412],
       [-1.33510889, -0.64613986],
       [-1.32702321,  0.6633044 ]])
经过PCA降维，数据特征个数是： 2
'''
```

9、SVD进行矩阵求逆

9.1、SVD求逆矩阵原理

在矩阵求逆过程中，矩阵通过 SVD 转换到正交空间。不同得奇异值和奇异值向量代表了矩阵中不同的线性无关

（或独立）项。对矩阵进行 SVD 分解，形式如下所示：

$$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n}$$

奇异值矩阵为：

$$\Sigma = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix}$$

当用 SVD 方法进行求逆时，会使得求逆运算变得非常简单，这是因为通过 SVD 求逆，只需要对奇异值求倒数即

可，而左奇异矩阵 U 和右奇异矩阵 V 都是正交矩阵，有 $U^{-1} = U^T$ ， $V^{-1} = V^T$ 。因此，其求逆形式为：

$$\begin{aligned} (A_{m\times n})^{-1} &= (U_{m\times m}\Sigma_{m\times n}V^T_{n\times n})^{-1} \\ &= (V^T_{n\times n})^{-1}(\Sigma_{n\times m})^{-1}(U_{m\times m})^{-1} \\ &= (V_{n\times n})(\Sigma_{n\times m})^{-1}(U^T_{m\times m}) \end{aligned}$$

奇异值矩阵逆矩阵为：

$$\Sigma^{-1} = \begin{bmatrix} \sigma_1^{-1} & & & \\ & \sigma_2^{-1} & & \\ & & \ddots & \\ & & & \sigma_n^{-1} \end{bmatrix}$$

从上面可以看出，SVD 求逆是原始奇异值的倒数，这就使得通过 SVD 对矩阵求逆变得非常简单：奇异值求倒数，

奇异矩阵转置。

9.2、SVD求逆代码演示

```
import numpy as np
A = np.array([[ 3,  4,  5,  5],
              [ 7,  5,  3,  6],
              [ 6,  5,  7,  7],
              [ 4,  9,  8,  9],
              [ 5, 10,  5,  7]])
# A是奇异矩阵
print('矩阵A的秩是：')
display(np.linalg.matrix_rank(A))
display(np.linalg.inv(A)) # 无法直接求解逆矩阵
'''
```

```
矩阵A的秩是：
4
报错信息：
LinAlgError: Last 2 dimensions of the array must be square
'''
```

使用奇异值分解，进行逆矩阵求解

```
import numpy as np
import warnings
warnings.filterwarnings('ignore')
A = np.array([[ 3,  4,  5,  5],
              [ 7,  5,  3,  6],
              [ 6,  5,  7,  7],
              [ 4,  9,  8,  9],
              [ 5, 10,  5,  7]])
u,s,v =np.linalg.svd(A)
display(u,s,v)
m,n = A.shape

# 奇异值求倒数
sigma = np.concatenate([np.diag(s),np.full(shape = (m-n,n),fill_value = 0)],axis
= 0)
sigma = sigma**(-1)
cond = np.isinf(sigma)
sigma[cond] = 0
# 逆矩阵求解
B = v.T.dot(sigma.T).dot(u.T)

print('矩阵B是A的逆矩阵，两个进行矩阵运算得到单位矩阵：')
display(B.dot(A).round(0))
'''
array([[-0.3101916 ,  0.04769317, -0.35708497, -0.01791768,  0.87958844],
       [-0.37495544, -0.73958189,  0.33372775,  0.44531378,  0.0524258 ],
       [-0.44812844, -0.34942375, -0.5206661 , -0.52493567, -0.36115552],
       [-0.55774569,  0.50103456, -0.20813073,  0.55345231, -0.29707954],
       [-0.50128858,  0.27858822,  0.66835958, -0.46851647,  0.06990107]])
array([27.56758049,  4.36911093,  3.79131357,  0.75187562])
array([[-0.39834686, -0.55822029, -0.46362907, -0.56103295],
       [-0.85451182,  0.46712883,  0.223151  , -0.04247162],
       [ 0.17147559,  0.64552498, -0.72590995, -0.16415961],
       [-0.28587663, -0.23125369, -0.45640603,  0.81024059]])
矩阵B是A的逆矩阵，两个进行矩阵运算得到单位矩阵：
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1., -0., -0.],
       [ 0., -0.,  1., -0.],
       [-0.,  0., -0.,  1.]])
'''
```

10、SVD进行协同过滤

10.1、协同过滤

协同过滤是一种从大量用户给出的兴趣爱好集合中预测用户的偏好和兴趣的技术。 它的**基本假设**是，对某个事物，人们过去喜欢那么将来也会喜欢。 协作过滤可以用于推荐系统，该推荐系统自动向用户建议他/她的首选产品。

推荐系统的任务就是联系用户和信息，一方面帮助用户发现对自己有价值的信息，而另一方面让信息能够展现在对它感兴趣的用户面前，从而实现信息消费者和信息生产者的双赢。

10.2、干饭人

下面是用户午餐点外卖倾向评分，分值1~5，分值越大表示倾向越大。0表示数据缺失，需要你使用SVD分解，将缺失值，进行补全，预测。数据如下

外卖意向	寿司	牛肉拉面	麻辣烫	黄焖鸡米饭	排骨饭
张三	2	1	3	2	5
李四	3	5	2	4	3
王五	1	3	1	1	4
赵六	0	2	4	5	2
Michael	4	4	5	0	5
Sara	5	5	2	3	1
John	1	1	4	2	2
Daniel	2	4	3	5	4
Po	1	3	3	2	1
Tom	5	0	1	3	5

奇异值分解算法可以用于矩阵近似问题。如果分解时，中间的矩阵取部分特征值（只取前面若干个最大的特征值），这样就可以对原矩阵进行近似了。基于这种思想，奇异值分解可以用于预测用户对外卖点餐的倾向评分。

10.3、SVD进行协同过滤

```
import numpy as np
food = np.mat([[2,1,3,2,5],
               [3,5,2,4,3],
               [1,3,1,1,4],
               [0,2,4,5,2],
               [4,4,5,0,5],
               [5,5,2,3,1],
               [1,1,4,2,2],
               [2,4,3,5,4],
               [1,3,3,2,1],
               [5,0,1,3,5]])
u, sigma, v = np.linalg.svd(food)
# 选取前2大特征值，做近似表达
food_result = np.mat(u[:, :2]) * np.mat(np.diag(sigma[:2])) * np.mat(v[:2, :])
```

```
food_result.round(1)
'''
array([[3.2, 1.9, 2.3, 1.5, 4.2],
       [2.5, 4.1, 3.5, 3.9, 3. ],
       [2.3, 1.8, 1.9, 1.5, 2.9],
       [1.2, 3.8, 2.9, 3.9, 1.3],
       [4.2, 2.8, 3.2, 2.3, 5.5],
       [2.3, 3.7, 3.2, 3.6, 2.8],
       [1.6, 2.2, 2. , 2.1, 2. ],
       [2.6, 4.3, 3.7, 4.2, 3.3],
       [1.1, 2.8, 2.2, 2.8, 1.2],
       [4. , 1.4, 2.2, 0.9, 5.3]])
'''
```

根据SVD分解，对缺失值数据，进行了预测，结果如下：

外卖意向	寿司	牛肉拉面	麻辣烫	黄焖鸡米饭	排骨饭
张三	2	1	3	2	5
李四	3	5	2	4	3
王五	1	3	1	1	4
赵六	1.2	2	4	5	2
Michael	4	4	5	2.3	5
Sara	5	5	2	3	1
John	1	1	4	2	2
Daniel	2	4	3	5	4
Po	1	3	3	2	1
Tom	5	1.4	1	3	5